PATENT ABSTRACTS OF JAPAN

(11)Publication number : 04-177461
(43)Date of publication of application : 24.06.1992

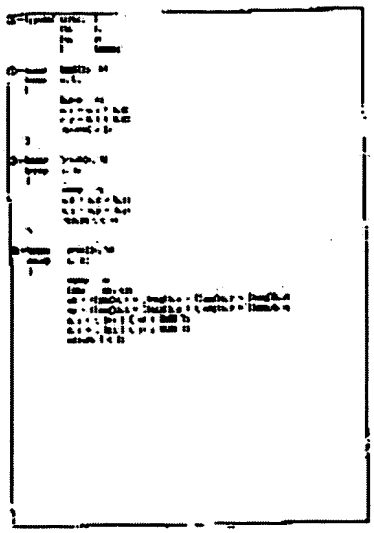
(51)Int.Cl. G06F 15/31
G06F 15/332

(21)Application number : 02-305035
(22)Date of filing : 07.11.1990

(71)Applicant : MITSUBISHI ELECTRIC CORP
(72)Inventor : KEGASA MITSUYOSHI

(54) HIGH SPEED TECHNICAL CALCULATION METHOD

(57)Abstract:
PURPOSE: To perform a technical calculation at a high speed without using any expensive numerical arithmetic processor and at the same time to reduce the necessary capacity of a memory byperforming a numerical operation through an integer arithmetic and in a block floating point format.
CONSTITUTION: A fixed floating point arithmetic using an integer arithmetic is expressed in a block floating point number that showed the data in an exponent part or a coefficient part common to all data and a fixed decimal point of each data. Then a block floating point arithmetic applying an integer arithmetic is carried out. It is supposed that a decimal, point is set at the 13th bit of an integer and then that 8192=213 of an integer is equivalent to 1.01 of a fixed decimal point. In this fixed decimal point format, the value covering −4.0 (minimum) through about 3.9999 (maximum) can be expressed. The expressible number most approximate to 0 is about 0.000122 and a number approximate to 1.0 has the decimal accuracy equivalent to about 4 digits. The diagram shows a program used for calculation of a complex number end applying a fixed decimal point in a C language.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

⑲日本国特許庁（JP）　⑪特許出願公開

⑫公開特許公報（A）　平4－177461

⑤1Int. Cl.⁵	識別記号	庁内整理番号	④3公開　平成4年(1992)6月24日
G 06 F 15/31	Z	6798－5L	
15/332	A	6798－5L	

審査請求　未請求　請求項の数 1　（全10頁）

⑤4発明の名称　高速技術計算方法

⑳特　願　平2－305035
㉒出　願　平2(1990)11月7日

⑦2発明者　毛笠　光容　兵庫県尼崎市塚口本町8丁目1番1号　三菱電機株式会社通信機制作所内
⑦1出願人　三菱電機株式会社　東京都千代田区丸の内2丁目2番3号
⑦4代理人　弁理士　大岩　増雄　外2名

明　細　書

1．発明の名称

高速技術計算方法

2．特許請求の範囲

構造化された汎用の高級言語上で、複数個のデータを処理する技術計算において、整数演算を利用したブロック化浮動小数点演算もしくは固定小数点演算を用いたことを特徴とする、高速技術計算方法。

3．発明の詳細な説明

〔産業上の利用分野〕

この発明は、コンピュータによる技術計算の高速化に関するものである。

〔従来の技術〕

第5図（a），（b）は、CQ出版の「Cによる科学技術計算」小池慎一著の209ページ～211ページに掲載されていたC言語による複素数演算プログラムである。

（1）は複素数を表現する形式complexを定義しており、複素数の実数部を倍精度浮動小数点（以降doubleと略す）のreであらわし、虚部をdoubleのimであらわしている。（2）は（1）で定義したcomplex型の加算用の関数caddであり、（3）はcomplex型の減算用の関数csul、（4）はcomplex型の乗算用の関数cmulであり、（5）はcomplex型の除算用の関数cdivである。（6）はcomplex型の数の絶対値を求める関数cabsであり、（7）はdouble型の数2つを各成分とするcomplex型の数を求める関数to-complexである。

第6図は同じく「Cによる科学技術計算」の218ページに掲載されていた複素数演算を用いたFFTプログラムである。（10）はサブルーチン名cfftの宣言部、（11）はビットリバーサル操作部の先頭、（12）はバタフライ積部の先頭を示している。

第7図（a），（b）は第6図を元にして図6のプログラムで省略されていたFFT演算後の割り算部を付加した実用的な複素数FFTプログラムである。（10）～（12）は第6図と同じもので

あり、（１３）はFFT演算後の配列Ｚの要素をｎで割る割り算部を示している。第８図は、ビットリバーサル部のフローチャート、第９図はバタフライ積部のフローチャート、第１０図は割り算部のフローチャートである。

次に動作について説明する。

第５図の（２）の関数caddは複素数の変数ａ，ｂを引き数として受け取り、その和である複素数Ｚを返すもので、

Z.re(Zの実部)＝a.re(aの実部)＋b.re(bの実部)

Z.im(Zの虚部)＝a.im(aの虚部)＋b.im(bの虚部)

を計算している。

第５図の（３）の関数csubは同様に複素数の差を返すもので、X.re＝a.re－b.re，X.im＝a.im－b.imを計算している。

第５図の（４）の関数cmulは同様に複素数の積を返すもので、

X.re＝a.re×b.re－a.im×b.im

X.im＝a.re×b.im－a.im×b.re

を計算している。

第５図の（５）の関数cdivは同様に複素数の商を返すもので、

$$X.re=\frac{a.re\times b.re+a.im\times b.im}{b.re\times b.re+b.im\times b.im}$$

$$X.im=\frac{a.im\times b.re+a.re\times b.im}{b.re\times b.re+b.im\times b.im}$$

を計算している。又、分母が０の時はエラー表示をするようになっている。

第５図の（６）の関数cabsはａの絶対値をdoubleで返す関数で$|a|=\sqrt{(a.re)^2+(a.im)^2}$を計算している。

第５図の（７）の関数tocomplexはdoubleの変数ｘとｙを引き数として受け取り、これを複素数Ｚに変換して返す関数でＺ.re＝ｘ，Ｚ.im＝ｙを実行している。

第６図，第７図は複素数演算を用いたFFTのプログラムの例である。

第８図は前記のプログラムのビットリバーサル部のフローチャートであり、第７図の（１１）の部分を説明している。

第９図は前記プログラムのバタフライ演算部のフローチャートであり、第７図の（１２）の部分を説明している。

第１０図は第７図の（１３）（第６図では省略されている）の割り算部のフローチャートである。

第７図のFFTプログラムcfftに複素数データの配列Ｚ，FFTの次数を表わす整数１，FFTとRFTを選択する整数の変数ｆ（ｆ＝１ならRFT，ｆ＝－１ならFFTが実行される）を引数として与えて実行させると、ｆ＝１なら配列Ｚのデータをフーリエ逆変換したデータが配列Ｚに求められる。ｆ＝－１なら、配列Ｚのデータをフーリエ変換したデータが配列Ｚに求められる。

FFTプログラムは、離数フーリエ変換を高速で行なうプログラムで時間的に変化する離数データｇ（ｋ）の周波数スペクトルＧ（ｎ）を、

$$G(n)=\frac{1}{N}\sum_{k=0}^{N-1}g(k)\,W_N^{nk}$$

という関係を用いて求めるプログラムである。但し、Ｎはデータ数で通常Ｚのベキ乗で表わされる数（2,4,8,･･･,128,256,512,1024,･･･）とし、W_Nは$e^{-i\frac{2\pi}{N}}$で表わされる数である。

FFTプログラムは主に電気信号や機械的信号などがどのような周波数成分を持つかを求めるのに使われており、古くは大型コンピュータでしか使われていなかったが、近年ではパソコン上や測定器上でも使われるようになっている。

第５図の（２）～（７）の複素数演算を使用して複素数FFTを記述すると、第７図のように短いプログラムとなる。

第７図のプログラムをパソコンPC9801RX2上でボーランド社のＣコンパイラTurbo-Cを用いて実行させると、データ数Ｎ＝1024の時、数値演算プロセッサがない場合で約１３秒必要であった。

数値演算プロセッサi80287-10を付けると、同じ演算が約３秒で実行できる。但しi80287-10は現状では４万円から６万円程度と高価である。

〔発明が解決しようとする課題〕

従来の技術計算方法は、以上のようにＣ言語で

サポートされている倍精度浮動小数点演算で行なわれていた。倍精度浮動小数点演算では各データを６４ビットの指数方式で表現している為、１０進数では１５桁もの高精度が得られるが、その反面、加減乗除等の基本的な演算でも整数演算の100～1000倍の時間を要し１データで８バイトのメモリを要する。この為、従来の技術計算プログラムでは長時間を要しかつ多くのメモリを必要とした。浮動小数点演算を高速化する為、数値演算プロセッサが存在するが、かなり高価である為、汎用のパーソナルコンピュータ等では実装されていないものが多い。

この発明は、上記のような問題点を解消するためになされたもので、高価な数値演算プロセッサを使わずに、短時間で結果を求めることができ、かつ必要なメモリの容量も少ない高速技術計算プログラムを得ることを目的とする。

〔課題を解決するための手段〕

この発明に係る高速技術計算方法は、データを固定小数点またはブロック浮動小数点で表わし、整数演算を利用した固定小数点演算もしくは、ブロック浮動小数点で計算を行なうものである。

〔作用〕

この発明における高速技術計算方法は、必要以上に高精度であり多くのメモリを必要とし、演算に長い時間を要する倍精度浮動小数点演算のかわりに整数演算を利用した固定小数点演算又は、データを全データ共通の指数部又は係数部と各データごとの固定小数点で表わしたブロック浮動小数点で表わし、整数演算を利用したブロックの浮動小数点演算で行なうことで技術計算を高速化しかつデータを格納するのに必要なメモリー容量を削減している。

〔実施例〕

以下この発明の一実施例を説明する。

本実施例では、固定小数点を整数を使って表現している。整数はコンピュータによって表現が異なるが、例えばパーソナルコンピュータPC9801シリーズ用のＣコンパイラTurbo-C ver2.0（ボーランド社）では、整数は１６ビットの符号付き整数型（以降intと略す）を標準としている。１６ビットの符号付き整数では2^{16}通りの数が表現でき最小-32768～最大32767が表現できる。

本実施例ではこの整数の１３ビット目に小数点があるものと仮定して整数の8192＝2^{13}が固定小数点の1.01に相当するものと設定した。この固定小数点フォーマットでは、最小-4.0～最大約3.9999が表現できる。表現できる０に最も近い数は約0.000122であり1.0付近の数なら１０進で約４桁分の精度がある。

この固定小数点での演算は、加減算なら整数と全く同じ演算で実行できるが、乗算では1.0×1.0=1.0となる様に調整する必要がある。又、１６ビットの整数の積は、最大約１０億となり１６ビット符号付整数では表現できなくなる。そこで乗算では、各整数を３２ビット符号付整数型（以降long intと略す）に直して計算する必要がある。３２ビット符号付整数では最小約－２億～最大約２１億までの数が表現できる。long intで乗算後8192で割り、int型に戻してやれば乗算ができる。例えば、固定小数点数a＝2.0,b＝0.5の積を固定小数点のＣに代入する演算をＣ言語で表記すると、

```
int a,b,c;
a=(int)(2.0*8192.0);
b=(int)(0.5*8192.0);
c=(int)((long int)a*(long int)b/8192);
```

となる。但し(int)はintへの型変換を指示するキャスト演算子である。又(long int)はlong intへの型変換を指示するキャスト演算子である。固定小数点の割り算も同様にlong intで計算する必要がある。例えばa＝1.0, b＝2.0でa割るbをcに代入するには

```
int a,b,c;
a=(int)(1.0*8192.0);b=(int)(2.0*8192.0);
c=(int)((long int)a*8192/(long int)b);
```

とＣ言語で記述できる。

このように固定小数点の演算を整数演算の組み合わせで実行することが可能である。

さらに複素数を２つの固定小数点数で表現する

ことも可能である。つまり複素数の実部を1つの固定小数点数で、虚部をもう1つの固定小数点数で表現することができる。これをC言語でbcomp型と定義すると以下のようになる。

```
typedef struct{int x; int y;}bcomp;
bcomp Z1,Z2,Z3;
```

この場合複素数Z1の実部がZ1.xに、虚部がZ1.yに対応する。Z1=0.7+j0.7, Z2=0.5+j0.8の和をZ3に代入するには上の定義式にて宣言した後

```
Z1.x=(int)(0.7*8192.0);
Z1.y=(int)(0.7*8192.0);
Z2.x=(int)(0.5*8192.0);
Z2.y=(int)(0.8*8192.0);
Z3.x=Z1.x+Z2.x;
Z3.y=Z1.y+Z2.y;
```

と書けば良い。ここで、わかるように浮動小数点型数から固定小数点数を利用した複素数への変換や、複素数の計算をそのまま記述すると大変見にくくかつ書くのも大変であるので、C言語の関数としてやることが好ましい。

第1図（a），（b）はC言語で固定小数点を利用した複素数計算用のプログラムである。最初の宣言部は上記説明のものと全く同じである。（2）の関数bcaddは加算用の関数であり、従来の実施例の関数caddのcomplexの部分をbcompに変えたものである。

（3）の関数bcsubは減算用の関数であり、これも従来の実施例の関数csubのcomplexの部分をbcompに変えたものである。

（4）の関数bcmulは積を求める関数であり、従来の実施例の関数cmulのcomplexの部分をbcompに変え、さらに実部，虚部各部の乗算を求める左辺で整数の型をintからlong intに変換してから乗算を行ない固定小数点に戻す為、8192で割ってから型をint型に戻すように書き変えている。

（5）の関数bcdivは商を求める関数であり、従来の実施例の関数cdivのcomplexをbcompに変え、整数の型変換を行なっている。固定小数点の商であるので被除数へ8192を乗じてから、除数で割れば良いのであるが、分子は2つの固定小数点数の積の和であり、例えば(2.0+j0.0)/(2.0+j0.0)を計算する場合分子の実部は整数で書けば$(16384)^2+0=268435456$であり、これに8192を乗ずると約2.2×10^{12}となりlong intでは表現できなくなる。long intの最大値は約2×10^9であり、先の例の268435456の8倍であるので商の計算ではたとえば、被除数及び除数の絶対値が2以下という条件付きで、分子は8倍までが可能である。そこで、分子8倍して分母は8192÷8=1024で割ってから割り算を実行し、その後long intからint型へ戻すように変更した。この為、除数の絶対値が整数で$\sqrt{1024}=32$固定小数点で0.004より小さいとエラーとなる。この時は、エラー表示をするようにした。

除算をさらに高速化するには、分子への積をやめて分母を8192で割れば良いが、この場合は除数の絶対値が整数で$\sqrt{8192}\fallingdotseq90$固定小数点で0.01より小さいとエラーとなる。

除算の精度を上げるには、たとえば被除数及び除数の絶対値を1以下という条件にしてやり、$2^{31}+(8192)^2=32$の余裕にして分子を32倍し、分母を256で割ってから割り算を行なえば良い。この場合、除数の絶対値が整数で$\sqrt{256}=16$，固定小数点で0.002より小さい時エラーとなる。

第1図において（6）は、複素数の絶対値を求める関数bcabsである。従来の実施例のcabsのcom-plexの部分をbcompにおき変えたものである。

第2図において（7）は、絶対値が2以下のdoubleの数2つをそれぞれ実部，虚部とするbcompの数を求める関数tobcompであり、従来の実施例のtocomplexとほぼ同じであるが、固定小数点フォーマットに直す為に8192を掛けている。

（8）は複素数bcompの数の配列Z〔 〕のn個の要素を2で割るサブルーチンbcnormであり、後で述べるFFTプログラムで使用している。

（9）は同じくbcompの配列Z〔 〕のn個1要素をその各成分（実部，虚部）の絶対値の最大が1以下になるように調整するサブルーチン

bcnorm2である。絶対値の最大値を１以下にする為にＺ〔　〕の全要素を変数constで割り、配列Ｚ〔　〕の共通の指数部に相当する変数ampをconst倍している。このサブルーチンでは、Ｚとampよりなるブロック浮動小数点フォーマットをサポートしている。

（１０）は、上で述べた複素演算を利用した複素FFTプログラムである。従来の実施例とほぼ同じアルゴリズムであるが、従来の実施例ではデータ数Ｎでの割り算をプログラムの最後でまとめて行なっていたが、本プログラムではバタフライ積演算を１組行なう度にデータをサブルーチンbcnormを使って２で割ることにより、Ｎでの割り算を実行している。FFTプログラムでは、バタフライ積を１組行なう度にデータの最大値が約２倍に増える性質があるので固定小数点が途中でオーバーフローすることがないように毎回２で割ることにより、Ｎでの割り算を実行させている。

この複素FFTプログラムのアルゴリズムの異なる部分を説明しているフローチャートを第１１図、第１２図に示す。第１１図において（１３）は追加されたbcnormサブルーチンの実行位置を示している。第９図の従来のプログラムでⒷで示すＮでの割り算プログラムへ移行するかわりに、第１１図ではサブルーチンを終了してメインプログラムへ戻っている。

図１２はbcnormサブルーチンのアルゴリズムを示すフローチャートで、配列Ｚの各要素を２で割っている。この様に固点小数点フォーマットの複素数bcmpを使って加減乗除算や、複素数FFTが実現できる。

第４図は、上記の複素数FFTプログラムをテストする為に作られたＣ言語のメインプログラムであり、1024個のデータの実部がサインカーブを描くよう設定し、虚部は全て０になるよう設定し、複素数FFTプログラムを実行している。

つまり、時間軸で正弦振動をしているようなデータを入力した時の周波数スペクトラムを求めいている。

pr_timeというサブルーチンは時刻を表示するプログラムでこのFFTプログラムで消費した時間を求めさせている。

このメインプログラムを実行させて時間を測った結果1024点のFFTが約２秒で完了した。

従来の実施例に比べて約６倍の高速化が実現できた。数値演算プロセッサを使用したものよりも少し遅い。但し精度は２桁～３桁程度である。例えば、８ビットのＡ／Ｄコンバータ等で入力されたデータの処理用としてはこの精度は充分な値であると思われる。又結果を画面（400ドット×640ドット）にグラフィック表示するようなアプリケーションプログラムでも、これだけの精度があれば十分である。

なお、上記実施例ではＣ言語上でプログラムを記述したがPASCAL等の他の構造化した高級言語でも同様のプログラムが作成できる。

又、上記実施例ではFFTを実行するプログラムを記述したが、同じような複数個のデータを処理する技術計算プログラムなら同様の手法で高速化が可能である。

又、上記実施例ではFFTを複素数演算を用いて計算しているが、複素数の実数部と虚数部のデータを分けてそれぞれ別の配列にして、通常の実数演算を用いて計算することも可能である。

又、上記実施例で使用したデータの複素数表現、複素数演算を用いて各種の技術計算を行なうことも可能であるが、技術計算の種類によっては固定小数点の精度が１０進数で約４桁である為、演算中に大きな誤差を生じる場合があるので注意が必要である。

〔発明の効果〕

以上のようにこの発明によれば、数値演算を整数演算を利用してブロック浮動小数点フォーマットで行なったので、高価な数値演算プロセッサを使うことなく、高速で技術計算を行なうことができ、かつ必要とするメモリ容量も削減することができる。

４．図面の簡単な説明

第１図はこの発明の一実施例による複素数の加減乗除と絶対値を求める関数の方法を示す図、第

2図はこの発明の一実施例による複素数の処理方法を示す図、第3図はこの発明の一実施例による複素FFTを示す図、第4図はこの発明の一実施例による図、第5図は従来の複素数演算用関数の図、第6図は従来の複素数演算を用いたFFT図、第7図は従来の実用的な複素数FFT図、第8図は従来及びこの発明の一実施例共通の複素数FFTプログラムのビットリバーサル部のフローチャート、第9図は従来の複素数FFTプログラムのバタフライ積部のフローチャート、第10図は従来の複素数FFTプログラムの割り算部のフローチャート、第11図はこの発明の一実施例による複素数FFTプログラムのバタフライ積部のフローチャート、第12図はこの発明の一実施例による複素数FFTプログラムのサブルーチンBCNORMのフローチャートである。

（1）は複素数の構造体宣言、（2）は複素数の加算用関数、（3）は複素数の減算用関数、（4）は複素数の乗算用関数、（5）は複素数の除算用関数、（6）は複素数の絶対値を求める関数、（7）は実数2つから複素数を作る関数、（8）は複素数の配列のn個の要素を2で割るサブルーチン、（9）は複素数の配列の要素の各成分の最大値を1より小さくするサブルーチン、（10）は複素数FFTサブルーチン、（11）はビットリバーサル部、（12）はバタフライ積部、（13）は割り算部を示す。

なお、図中、同一符号は同一もしくは相当部分を示す。

代理人　大岩増雄

```
①-(typedef struct {
        int     x;
        int     y;
        }       bcomp;

②-bcomp  bcadd(a, b)
   bcomp  a, b;
   {
        bcomp   c;
        c.x = a.x + b.x;
        c.y = a.y + b.y;
        return( c );
   }

③-bcomp  bcsub(a, b)
   bcomp  a, b;
   {
        bcomp   c;
        c.x = a.x - b.x;
        c.y = a.y - b.y;
        return ( c );
   }

④-bcomp   bcmul(a, b)
   bcomp   a, b;
   {
        bcomp   c;
        long    xx, xy;
        xx = (long)a.x * (long)b.x - (long)a.y * (long)b.y;
        xy = (long)a.x * (long)b.y + (long)a.y * (long)b.x;
        c.x = ( int ) ( xx / 8192 );
        c.y = ( int ) ( xy / 8192 );
        return ( c );
   }
```

第1図(a)

```
⑤-bcomp   bcdiv(a, b)
   bcomp   a, b;
   {
        bcomp   c;
        long    z, zx, zy;
        z = (long)b.x * (long)b.x + (long)b.y * (long)b.y;
        z /= 1024;
        if ( z == 0 ) fprintf(stderr,"¥n Divide by 0 error in bcdiv ¥n");
        zx = (long)a.x * (long)b.x + (long)a.y * (long)b.y;
        zy = (long)a.y * (long)b.x - (long)a.x * (long)b.y;
        c.x = ( int ) ((zx / z)*8);
        c.y = ( int ) ((zy / z)*8);
   }
        return ( c );

⑥-double  bcabs( a )
   bcomp   a;
   {
        double x,y,z;

        x = (double)a.x;
        y = (double)a.y;
        z = sqrt( x * x + y * y ) / 8192.0;
        return ( z );
   }
```

第1図(b)

```
dcomp   tobcomp(a, b)
double  a, b;
{
        bcomp   c;
        c.x = (int) ( a * 8192.0 );
        c.y = (int) ( b * 8192.0 );
        return ( c );
}

void    bcmove(z, n)
bcomp   *z;
int     n;
{
        int     i;
        for( i = 0 ; i < n ; i++ )
        {
                (*z).x /= 2;
                (*z).y /= 2;
                z++;
        }
}
```

第 2 図(a)

```
void    bcmove2(z, n, exp)
bcomp   *z;
int     n, exp;
{
        int     maximum,i, const;
        bcomp   *p;
        p = z;
        maximum = 0;
        for ( i = 0 ; i < n ; i++ )
        {
                if ( abs((*p).x) > maximum) maximum = abs((*p).x);
                if ( abs((*p).y) > maximum) maximum = abs((*p).y);
        }
        const = 1;
        while ( maximum > 8192)
        {
                maximum /= 2;
                exp += 2;
                const *= 2;
        }
        if ( const == 1 ) return;
        p = z;
        for ( i = 0 ; i < n ; i++ )
        {
                (*p).x /= const;
                (*p).y /= const;
        }
}
```

第 2 図(b)

```
void    cfft(z,l,f)
bcomp   *z;
int     l,f;
{
        int     n,nv2,lp,i,j,k,m,le,lei;
        bcomp   t,w,u;
        double  c, s;

        n = (int)pow(2.0,(double)l);
        nv2 = n / 2;

        for (i=0,j=0; i<n-1 ; i++)
        {
                if( i<j )
                {
                        t = z[i];
                        z[i] = z[j];
                        z[j] = t;
                }
                k =nv2;
                while ( k<=j )
                {
                        j -= k;
                        k /= 2;
                }
                j += k;
        }

        for ( m=1; m<=l ; m++ )
        {
                le = (int)pow(2.0,(double)m);
                lei = le / 2;
                u = tobcomp(1.0, 0.0);
                c = cos( M_PI / (double) lei );
                s = ( (double) f ) * sin( M_PI / (double) lei );
                w = tobcomp( c, s );
                for (j=0; j<le/2 ;j++)
                {
                        for ( i=j; i < n ; i += le )
                        {
                                lp = i + lei;
                                t = bcmul(z[lp],u);
                                z[lp] = bcsub(z[i],t);
                                z[i] = bcadd(z[i],t);
                        }
                        u = bcmul( u, w );
                }

                bcmove( z, n );
        }
}
```

第 3 図

```
void    main(void)
{
double  dx, x, y;
int     i,l,f;
bcomp   z[1024];
        dx = 2.0 * M_PI / 1024.0;
        x = 0;
        for( i=0 ; i < 1024 ; i++ )
        {
                z[i] = tobcomp( sin(x),0.0 );
                x += dx;
        }
        printf(" Time domain \n");
        for( i=0 ; i < 8 ; i++ )
        {
                x = ( (double)z[i].x ) / 8192.0;
                printf( " z[%2d].x = %lf \n", i, x );
        }
        pr_time();
        cfft( z, 10, -1 );
        pr_time();
        printf(" Frequency domain \n");
        for( i=0 ; i < 8 ; i++ )
        {
                x = ( (double)z[i].x ) / 8192.0;
                y = ( (double)z[i].y ) / 8192.0;
                printf( " z[%2d].x = %lf, z[%2d].y = %lf \n", i, x, i, y);
        }
}
```

第 4 図

```
① typedef struct cplx
        {       double re;
                double im;
        } complex;

  complex cadd(),csub(),cmul(),cdiv(),conj(),corp(),cpow(),
        crt(),tocomplex(),clap(),
  double cabs(),carg();

② complex cadd(a, b)
  complex a, b;
  {
        complex z;

        z.re=a.re + b.re; z.im=a.im + b.im;
        return z;
  }

③ complex csub(a,b)
  complex a, b;
  {
        complex z;

        z.re=a.re - b.re; z.im=a.im - b.im;
        return z;
  }

④ complex cmul(a,b)
  complex a, b;
  {
        complex z;

        z.re=a.re*b.re-a.im*b.im;
        z.im=a.re*b.im+a.im*b.re;
        return z;
  }
```

第 5 図 (a)

```
⑤ complex cdiv(a,b)
  complex a, b;
  {
        complex z;
        double d;

        d=b.re * b.re+b.im*b.im;
        if ( d==0 )
        {
                printf("\7\7\7divided by zero error!!!\n");
                d = TINY;
        }
        z.re=(a.re*b.re+a.im*b.im)/d;
        z.im=(a.im*b.re-a.re*b.im)/d;
        return z;
  }

⑥ double cabs(a)
  complex a;
  {
        return (sqrt(a.re*a.re+a.im*a.im));
  }

  double carg(a)
  complex a;
  {
        if ( (a.re==0) && (a.im==0) )
        {
                printf("\7\7\7 arg zero error!!!\n");
                return 10.0;
        }
        else    return (atan2(a.im,a.re));

  }

  complex  conj(a)
  complex  a;
  {
        complex z;

        z.re=a.re; z.im=-a.im;
        return z;
  }

⑦ complex  tocomplex(x,y)
  double x,y;
  {
        complex z;
        z.re=x; z.im=y;
        return z;
  }
```

第 6 図 (b)

```
⑧ int cfft( z, l, f )     /* FFT by using complex type                    */
  complex  *z;            /* adapted by S.Saito from Ref(3)               */
  double   f;             /*  f=-1.0,... fft                              */
  int      l;             /*     1.0,... rft                              */
  {
        int     n, nv2, ip, i, j, k, m, le, le1;
        complex t, w, u;
        double a, c ;

        n = ipow2(l);  nv2=n/2 ;

⑨ for (i=0,j=0; i<n-1; i++)              /* bit reversal operation */
   {
        if ( i<j )
        {
                t=z[i]; z[i]=z[j]; z[j]=t;
        }
        k=nv2;
        while ( k<=j )
        {
                j-=k; k/=2;
        }
        j+=k;
   }

   /***** main loop *****/

⑩ for ( m=1; m<=l; m++)
   {
        le= ipow2(m); le1=le/2;
        w = tocomplex(1.0, 0.0);
        c = cos( PI/(double)le1 );
        a = -sin(PI/(double)le1);
        u = tocomplex( c, a );
        for (j=0; j<le/2; j++)
        {
                for (i=j; i<n; i+=le)
                {
                        ip=i+le1;
                        t =cmul(z[ip],w); z[ip]=csub(z[i],t);
                        z[i]=cadd(z[i],t);
                }
                w=cmul( u, w);
        }
   }
  }
```

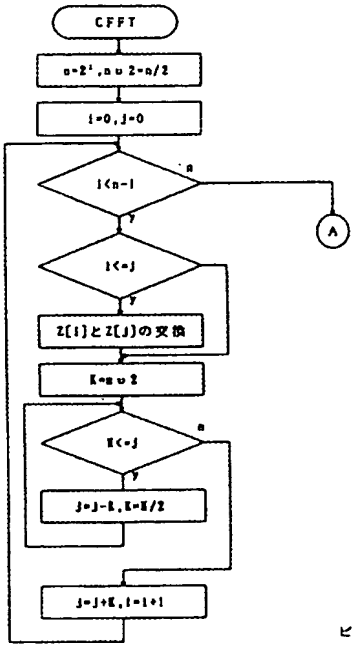
第 8 図

```
  #include <stdio.h>
  #include <math.h>
  #include <complex.h>

⑪ void    cfft(z,l,f)
  complex *z;
  int     l,f;
  {
        int     n, nv2, ip, i, j, k, m, le, le1;
        complex t, w, u;
        double  a, c;

        n = (int)pow(2.0,(double)l);
        nv2 = n / 2;

⑫ for (i=0, j=0; i<n-1 ; i++)
   {
        if( i<j )
        {
                t = z[i];
                z[i] = z[j];
                z[j] = t;
        }
        k = nv2;
        while ( k<=j )
        {
                j -= k;
                k /= 2;
        }
        j += k;
   }
```

第 7 図 (a)

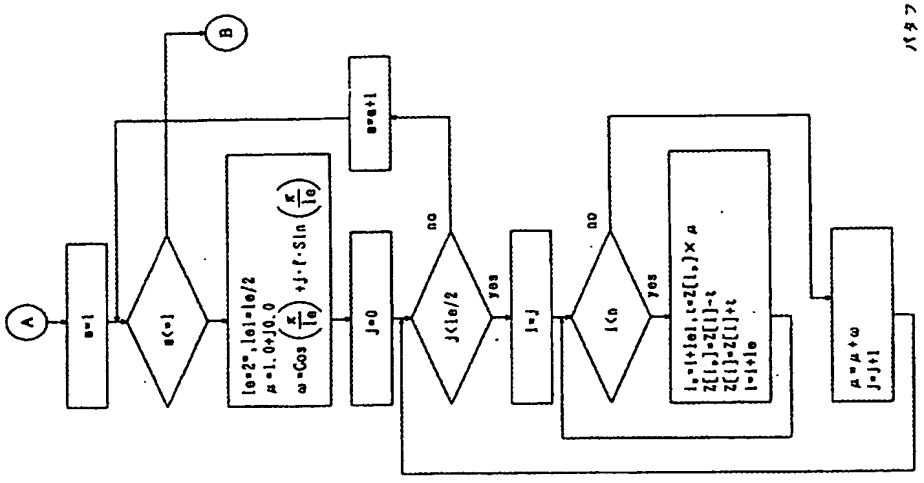
```
⑫ -for ( n=1; n<=l ; n++ )
   {
        le = (int)pow(2.0,(double)n);
        lei = le / 2;
        u = tocomplex(1.0, 0.0);
        c = cos( M_PI / (double) lei );
        s = ( (double) f ) * sin( M_PI / (double) lei );
        w = tocomplex( c, s );
        for (j=0; j<le/2 ; j++)
        {
             for ( i=j; i < n ; i += le )
             {
                  ip = i + lei;
                  t = cmul(z[ip],u);
                  z[ip] = csub(z[i],t);
                  z[i] = cadd(z[i],t);
             }
             u = cmul( u, w );
        }
   }

⑬ -if ( f < 0 ) for ( i = 0 ; i < n ; i++ )
   {
        z[i].x /= (double) n;
        z[i].y /= (double) n;
   }
}
```

第7図(b)



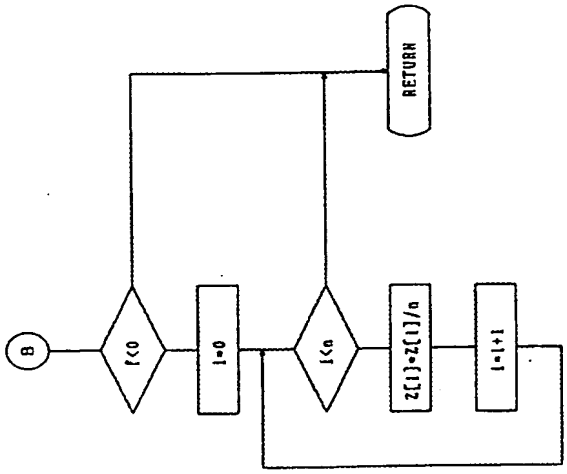
ビットリバーサル部

第8図



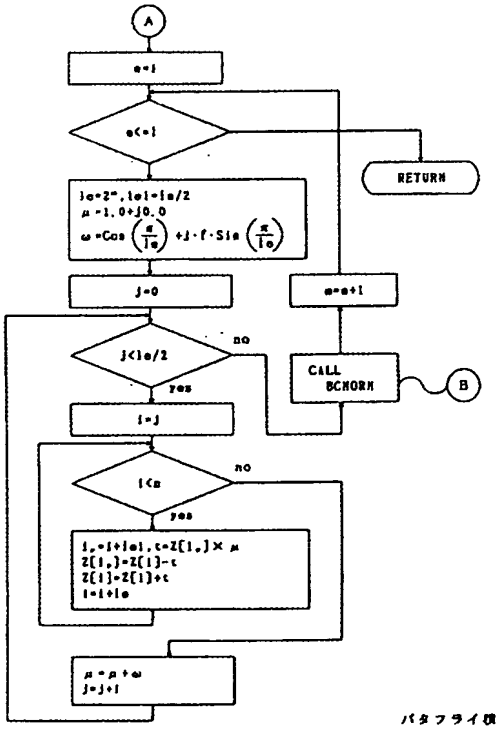
バタフライ演算部

第9図



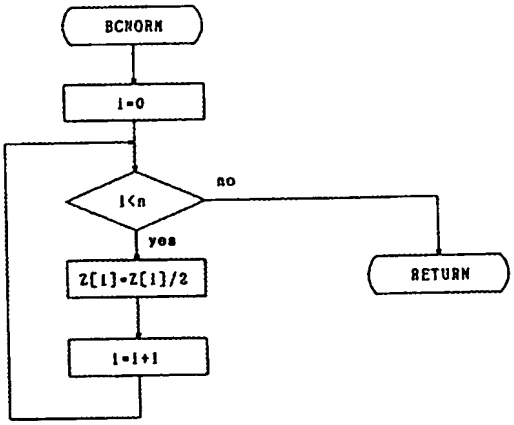
FFT時の割り算部

第10図



バタフライ演算部

第11図



第12図